UNIVERSITAS INDONESIA

PENERAPAN METODE MEDIAN FILTER UNTUK PENGURANGAN NOISE PADA CITRA DIGITAL

LAPORAN UJIAN AKHIR SEMESTER

KOMPUTASI LANJUT DAN BIG DATA

AGATHA ULINA SILALAHI

2306288875

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

PROGRAM STUDI MAGISTER MATEMATIKA

DEPOK

2023

BAB 1
PENDAHULUAN

1.1. Latar Belakang

Pemrosesan citra adalah suatu bidang ilmu yang bertujuan untuk memanipulasi gambar atau citra dengan maksud tertentu. Citra dapat berupa foto, gambar medis, peta, atau jenis visual lainnya. Pemrosesan citra digital telah menjadi bidang penelitian dan pengembangan yang semakin penting dalam dunia modern, dengan penerapan yang meluas di berbagai sektor seperti kedokteran, industri, keamanan, dan komunikasi. Dengan nilai piksel yang terus meningkat, pemrosesan citra tidak hanya menjadi kunci dalam fotografi dan industri kreatif, tetapi juga memiliki aplikasi luas dalam berbagai disiplin ilmu, termasuk pengenalan pola, kecerdasan buatan, dan diagnostik medis.

Dalam era di mana informasi visual mendominasi, pemrosesan citra menjadi sangat penting untuk mengekstrak, menganalisis, dan menginterpretasi data visual. Kemampuan manusia untuk memahami dan membuat keputusan seringkali didasarkan pada informasi visual, dan pemrosesan citra memungkinkan komputer untuk melakukan hal serupa. Peningkatan kemampuan komputasi, perangkat keras yang lebih canggih, dan algoritma pemrosesan citra yang inovatif memungkinkan pengembangan aplikasi yang lebih maju dan kompleks. Dengan semakin meningkatnya kompleksitas tugas pemrosesan citra dan peningkatan kebutuhan dalam aplikasi dunia nyata, penelitian dan pengembangan di bidang ini terus berlanjut. Penerapan kecerdasan buatan dan teknologi terkini lainnya menjadikan pemrosesan citra sebagai bidang yang dinamis dan penting dalam perkembangan teknologi informasi.

Beberapa contoh implementasi pemrosesan citra adalah grayscale conversion, thresholding, dan negative transformation. Grayscale conversion merupakan suatu teknik penting dalam dunia pemrosesan gambar yang memainkan peran krusial dalam mengubah representasi warna menjadi skala abu-abu. Dalam konteks ini, warna pada gambar dikonversi menjadi tingkat keabu-abuan, tanpa mempertahankan informasi warna asli. Grayscale conversion bukan hanya sekadar menghilangkan dimensi warna pada gambar, namun juga memberikan fokus yang lebih kuat pada intensitas cahaya dan bayangan. Hal ini memungkinkan analisis lebih mendalam terhadap struktur dan pola pada gambar, memudahkan interpretasi visual dan ekstraksi fitur penting. Dalam beberapa kasus, operasi pada gambar berwarna lebih kompleks dan membutuhkan lebih banyak sumber daya. Grayscale conversion dapat meningkatkan efisiensi pemrosesan, terutama ketika warna tidak diperlukan.

Kemudian, thresholding adalah teknik penting dalam domain pemrosesan gambar yang memainkan peran utama dalam mengonversi gambar menjadi format biner. Tujuan utama dari thresholding adalah untuk membedakan piksel-piksel gambar menjadi dua kategori, yaitu hitam dan putih, berdasarkan suatu nilai batas tertentu. Teknik ini sangat berguna dalam mengidentifikasi objek atau fitur tertentu dalam gambar yang memiliki perbedaan intensitas cahaya yang signifikan. Dengan menentukan ambang batas yang tepat, piksel-piksel yang memiliki intensitas cahaya di atas batas akan dianggap putih, sementara piksel dengan intensitas di bawah batas akan dianggap hitam. Hasilnya adalah gambar biner yang memudahkan ekstraksi informasi, segmentasi objek, dan analisis fitur. Selanjutnya, negative transformation adalah teknik dalam pemrosesan gambar yang melibatkan pembalikan intensitas piksel pada suatu gambar. Dalam konteks ini, nilai intensitas cahaya pada setiap piksel diubah sedemikian rupa sehingga piksel yang semula terang menjadi gelap, dan sebaliknya. Tujuan utama dari negative transformation adalah menciptakan efek visual yang kontras, menghasilkan gambar yang terkesan invers dari gambar aslinya.

Pengambilan citra atau gambar pada saat menggunakan kamera sering mengalami derau atau noise pada citra yang disebabkan oleh lensa kamera yang buruk ataupun posisi kamera saat pengambilan citra sehingga dapat menyebabkan terjadinya derau (noise) pada citra yang dihasilkan. Noise pada citra tersebut pada umumnya terdistribusi secara normal (Gaussian). Pengaruh noise pada citra sangat mempengaruhi kualitas citra sehingga noise pada citra harus dihilangkan agar kualitas jauh lebih baik dari sebelumnya dimana kualitas citra merupakan salah satu proses awal dalam pengolahan citra (image preprocessing). Perbaikan kualitas diperlukan karena seringkali citra yang dijadikan objek pembahasan mempunyai kualitas yang buruk, misalnya citra mengalami derau (noise) pada saat pengiriman melalui saluran transmisi, citra terlalu terang/gelap, citra kurang tajam, kabur, dan sebagainya. Melalui operasi pemrosesan awal inilah kualitas citra diperbaiki sehingga citra dapat digunakan untuk aplikasi lebih lanjut, misalnya untuk aplikasi pengenalan (recognition) objek di dalam citra. Yang dimaksud dengan perbaikan kualitas citra adalah proses mendapatkan citra yang lebih mudah diinterpretasikan oleh mata manusia. Pada proses ini, ciri-ciri tertentu yang terdapat di dalam citra lebih diperjelas kemunculannya. Perbaikan citra bertujuan meningkatkan kualitas tampilan citra untuk pandangan manusia atau mengkonversi suatu citra agar memiliki format yang lebih baik sehingga citra tersebut menjadi lebih mudah diolah dengan mesin (komputer). Pengurangan noise atau denois adalah salah satu proses dalam perbaikan kualitas citra (image enhancement) yang termasuk langkah awal dalam image processing. Perbaikan kualitas citra adalah suatu teknik yang memperhatikan bagaimana mengurangi perubahan bentuk dan

penurunan kualitas citra yang diawali selama pembentukan citra tersebut. Gonzales dan Wood mendefinisikan restorasi citra sebagai proses sebagai proses yang berusaha merekonstruksi atau mengembalikan suatu citra yang mengalami degradasi.

1.2. Rumusan Masalah

Dengan berdasarkan latar belakang yang telah dikemukakan diatas, maka yang menjadi pokok permasalahan dalam hal ini yaitu bagaimana menerapkan metode median filter untuk menghilangkan impulse noise pada citra dengan mempertahankan detail objek pada citra.

1.3. Tujuan Penelitian

Tujuan dari project ini adalah untuk menganalisa performa Median Filter, kombinasi dan rekursif filter untuk menghilangkan impulse noise dan additive noise, yang dilakukan secara kualitatif dengan membandingkan citra asli, citra terkena noise dan citra output dari filter, dan secara kuantitatif dengan membandingkan nilai PSNR-nya. Efek noise diberikan terhadap gambar asli yang bersih (clean image) dengan meng- gunakan fungsi noise yang tersedia pada python

BAB 2
TINJAUAN PUSTAKA

2.1. Image Processing

Image processing atau pengolahan gambar adalah sebuah metode untuk mengoperasikan gambar dengan tujuan meningkatkan kualitas gambar atau mengekstraksi sejumlah informasi berguna dari gambar tersebut. Pengolahan gambar merupakan bidang yang penting di dunia komputasi dilihat dari banyaknya riset mengenai pengolahan gambar. Secara umum, pengolahan gambar meliputi tiga langkah berikut (Unversity of Tartu, 2014):

1. Mengakuisisi gambar melalui kakas akuisisi,
2. Menganalisa dan memanipulasi gambar,
3. Mengembalikan keluaran berupa gambar yang telah dimanipulasi atau laporan berdasarkan analisis gambar

Menurut Jain (1989), pengolahan gambar dilakukan jika kondisi-kondisi berikut terpenuhi:

1. Peningkatan kualitas tampilan gambar atau menonjolkan beberapa aspek informasi yang terkandung di dalam gambar.
2. Elemen di dalam gambar perlu dikelompokkan. dicocokkan, atau diukur,
3. Sebagian gambar perlu digabungkan dengan bagian gambar yang lain.

Pengolahan gambar dapat dilakukan untuk gambar analog atau gambar digital. Pengolahan gambar analog dilakukan dengan memanipulasi sinyal analog dua dimensi pada gambar, seperti gambar pada televisi. Pengolahan gambar digital dilakukan dengan memanipulasi piksel-piksel gambar melalui perangkat lunak komputer (Unversity of Tartu, 2014). Pengolahan gambar digital mempunyai keuntungan dibandingkan pengolahan gambar analog karena lebih banyaknya operasi-operasi yang dapat dilakukan.

Gonzalez & Woods (2008) mendefinisikan proses-proses pengolahan gambar digital sebagai berikut:

1. Peningkatan kualitas gambar (image enhancement). Image enhancement adalah proses memanipulasi gambar sehingga gambar hasil manipulasi lebih cocok untuk digunakan pada keperluan tertentu dibandingkan gambar sebenarnya. Setiap keperluan membutuhkan enhancement yang berbeda, yakni enchancement untuk gambar X-Ray berbeda dengan enhancement dengan gambar dari satelit. Image enhancement bersifat subjektif karena pengguna image enhancement yang menentukan seberapa bagus sebuah teknik dilakukan.
2. Pemulihan gambar (image restoration). Image restoration adalah proses yang juga bertujuan untuk meningkatkan penampilan sebuah gambar. Akan tetapi, image

restoration berbasis perhitungan matematika atau model probabilistik sehingga peningkatan dari image restoration bersifat objektif.

3. Pemampatan gambar (image compression). Image compression adalah proses mengurangi besar memori yang dibutuhkan untuk menyimpan gambar atau mengurangi bandwidth yang dibutuhkan untuk Mengirimkan gambar tersebut. Image compression banyak dikenal oleh masyarakat luas dalam bentuk ekstensi file gambar, seperti jpg, png, dan lainnya.

2.2. Image Enhancement

Peningkatan kualitas gambar (image enhancement). Image enhancement adalah proses memanipulasi gambar sehingga gambar hasil manipulasi lebih cocok untuk digunakan pada keperluan tertentu dibandingkan gambar sebenarnya. Setiap keperluan membutuhkan enhancement yang berbeda, yakni enhancement untuk gambar X-Ray berbeda dengan enhancement dengan gambar dari satelit (Gonzalez & Woods, 2008).

Image enhancement dapat dilakukan dengan menggunakan berbagai metode, yakni transformasi intensitas, pemrosesan histogram, dan filter spasial. Transformasi intensitas terbagi menjadi tiga yakni linear, logaritmik, dan power-law. Pemrosesan histogram terbagi menjadi perataan histogram dan spesifikasi histogram. Filter spasial terbagi menjadi filter untuk pelembutan, filter untuk penajaman, dan filter kombinasi. Filter spasial tidak dibahas karena diluar scope tulisan ini.

2.2.1 Transformasi Intensitas

Transformasi intensitas secara linear terbagi menjadi transformasi identitas dan transformasi negatif. Transformasi identitas tidak mengubah apapun dari gambar sebenernya. Transformasi negatif bertujuan untuk membalikkan level intensitas sebuah gambar sehingga keluaran transformasi ekivalen dengan negatif secara fotografi. Transformasi negatif banyak digunakan memperjelas warna putih atau abu-abu di dalam sebuah gambar berlatar gelap. Jika r adalah nilai piksel sebelum transformasi, s adalah nilai piksel setelah transformasi, dan L adalah nilai maksimum piksel maka:

$$s = L - 1 - r$$

dimana,

s: Nilai intensitas pixel pada citra output

$L - 1$: Nilai intensitas maksimum (untuk grayscale adalah 255)

r: Nilai intensitas pixel pada citra input

Transformasi intensitas secara logaritmik terbagi menjadi transformasi log dan transformasi inverse-log. Transformasi log bermanfaat untuk memperluas nilai dari piksel-piksel gelap sembari menurunkan nilai piksel-piksel terang. Transformasi inverse-log melakukan kebalikan dari transformasi log. Jika r adalah nilai piksel sebelum transformasi, s adalah nilai piksel setelah transformasi, dan c adalah konstanta maka:

$$s = c.log(1 + r)$$

$$c = \frac{255}{\log(\max(img) + 1)}$$

dimana,

c: konstanta

r: Nilai intensitas pixel pada citra input

Transformasi intensitas secara power-law atau transformasi gamma mempunyai persamaan sebagai berikut:

$$s = c.r^{\gamma}$$

Transformasi gamma dengan nilai γ yang rendah akan memperluas piksel-piksel bernilai gelap sedangkan hal yang berlawanan terjadi jika nilai γ tinggi. Transformasi linear identitas dapat dicapai jika γ bernilai satu.

2.3. Grayscale

Gambar terbentuk dari sejumlah piksel dan berbagai parameter utama seperti warna dan monokrom (kadang-kadang disebut sebagai gambar hitam-putih atau propertinya). Gambar diproses dan dieksekusi melalui teknik pemrosesan gambar. Konversi grayscale juga merupakan bagian penting dari pemrosesan gambar. Informasi RGB atau warna memiliki sifat tiga dimensi yang membuat pemrosesan sinyal menjadi begitu besar dan berat, untuk mengatasi kelemahan ini, konversi grayscale menjadi suatu keharusan. Gambar grayscale adalah gambar di mana informasi warna hilang dan semua informasi warna dikonversi menjadi format grayscale. Gambar grayscale berbeda dari gambar hitam putih bitonal satu bit, yang dalam konteks citra komputer adalah gambar dengan hanya dua warna, hitam dan putih. Gambar grayscale memiliki banyak nuansa abu di antaranya.

Dalam bagian ini, kami secara singkat menggambarkan metode dengan kompleksitas waktu linear untuk mengonversi dari warna ke grayscale, yaitu fungsi G yang mengambil gambar warna $R^{n\times m\times 3}$ dan mengonversinya ke representasi $R^{n\times m}$. Semua nilai gambar

diasumsikan berada antara 0 dan 1. Mari R, G, dan B mewakili saluran merah, hijau, dan biru secara linear (artinya, tanpa koreksi gamma). Keluaran dari setiap algoritma skala abu-abu berada di antara 0 dan 1. Semua transformasi diterapkan komponen-wise, yaitu diterapkan secara independen pada setiap piksel. Beberapa metode menggunakan fungsi koreksi gamma standar $\Gamma(t) = t' = t^{\frac{1}{2.2}}$ dan ditunjukkan saluran yang sudah dikoreksi gamma sebagai R', G', dan B'. Mungkin algoritma paling sederhana untuk mengonversi warna ke skala abu-abu adalah Intensity. Ini adalah rata-rata dari saluran RGB :

$$G_{Intensify} \leftarrow \frac{1}{3}(R + G + B)$$

Meskipun Intensity dihitung menggunakan saluran linear, dalam praktiknya koreksi gamma seringkali dibiarkan utuh ketika menggunakan dataset yang berisi gambar yang sudah dikoreksi gamma. Kami menyebut metode ini sebagai Gleam :

$$G_{Gleam} \leftarrow \frac{1}{3}(R' + G' + B')$$

Dalam hal nilai piksel, Intensity dan Gleam menghasilkan hasil yang sangat berbeda. Karena CðtÞ adalah fungsi konkaf, *Jensen's inequality* menyiratkan bahwa Gleam tidak akan pernah menghasilkan representasi dengan nilai yang lebih besar dari Intensity yang sudah dikoreksi gamma, dan dari sini dapat disimpulkan bahwa

$$G_{Intensify} \leq G_{Gleam} \leq \Gamma\left(G_{Intensify}\right)$$

Ketika Intensity yang sudah dikoreksi gamma dan Gleam diterapkan pada gambar alami, kami menemukan bahwa Gleam menghasilkan nilai piksel yang rata-rata lebih kecil sekitar 20-25%.

2.4. Thresholding

Thresholding merupakan pendekatan mendasar dalam segmentasi gambar. Ini melibatkan membagi gambar menjadi berbagai wilayah berdasarkan nilai ambang tertentu yang dipilih. Makalah membahas berbagai teknik thresholding seperti single-thresholding dan multiple-thresholding, serta adaptive thresholding, optimal thresholding, dan local thresholding. Single-thresholding, misalnya, melibatkan pemilihan tingkat ambang untuk mengategorikan piksel berdasarkan nilai kecerahan mereka, sementara multiple-thresholding berurusan dengan histogram multi-modal dengan menggunakan beberapa nilai ambang. Ambang yang dipilih menentukan bagaimana gambar di-segmentasi ke dalam berbagai wilayah, yang penting untuk analisis dan interpretasi lebih lanjut dari gambar.

Thresholding juga merupakan teknik dalam OpenCV dikenal sebagai teknik penugasan nilai piksel berdasarkan nilai ambang yang disediakan. Dalam proses penambangan ambang,

setiap nilai piksel dibandingkan dengan nilai ambang. Jika nilai piksel lebih kecil dari ambang, maka nilainya diatur menjadi 0, sebaliknya, jika lebih besar, nilainya diatur menjadi nilai maksimum (biasanya 255). Penambangan ambang merupakan teknik segmentasi yang sangat populer, digunakan untuk memisahkan objek yang dianggap sebagai foreground dari latar belakangnya. Ambang adalah nilai yang memiliki dua wilayah di kedua sisinya, yaitu di bawah ambang atau di atas ambang. Dalam Computer Vision, teknik penambangan ambang dilakukan pada gambar grayscale. Jadi, pada awalnya, gambar harus dikonversi ke dalam ruang warna grayscale.

If f (x, y) < T
then f (x, y) = 0
else
f (x, y) = 255
where
f (x, y) = Coordinate Pixel Value
T = Threshold Value.

Dalam OpenCV dengan Python, fungsi “cv2.threshold” digunakan untuk proses thresholding.
Syntax: cv2.threshold(source, thresholdValue, maxVal, thresholdingTechnique)
Parameters:
-> source: Input Image array (must be in Grayscale).
-> thresholdValue: Value of Threshold below and above which pixel values will change accordingly.
-> maxVal: Maximum value that can be assigned to a pixel.
-> thresholdingTechnique: The type of thresholding to be applied.

2.5. OpenCV

OpenCV (Open Computer Vision) adalah sebuah library yang mengkhususkan dirinya untuk pembangunan computer vision dan machine learning. OpenCV sendiri sudah memiliki sangat banyak algoritma yang teroptimasi dengan jumlah lebih dari 2500. Library OpenCV sendiri tidak hanya ada untuk bahasa pemrograman Python saja, melainkan ada pula untuk bahasa lain seperti C++, Java, dan MATLAB Interfaces dan sudah sangat support penggunaannya pada operasi sistem Windows, Linux, Mac OS dan Android.
Fungsi-Fungsi Utama OpenCV

Kelas Input Output

- Kelas imread

Imread merupakan akronim dari “Image Read” yang berarti membaca citra. Sesuai dengan artinya perintah ini digunakan untuk mengambil nilai matrik pada citra digital kemudian akan disimpan dalam objek Mat. Sehingga nilai matrik dari citra digital dapat diolah sesuai dengan kebutuhan. Class ini membutuhkan input lokasi citra dan variable flag.

- Kelas imwrite

Kebalikan dari imread perintah imwrite memiliki akronim “Image write” yang berarti menulis citra. Perintah ini biasa digunakan untuk membuat suatu citra digital yang berasal dari suatu objek Mat.

- Kelas Mat

Kelas Mat ini merupakan tipe data yang dapat digunakan untuk menyimpan n dimensi data (Bradski dan Kaehler, 2008). Kelas ini biasa digunakan untuk menampung nilai vektor atau matriks. Kelas ini biasanya digunakan untuk menampung nilai matrik suatu gambar. Untuk gambar berwarna, matrik akan disimpan dalam 3 kanal secara otomatis, dimana tiap kanal akan mewakili matriks nilai dari warna merah, hijau dan biru. Sedangkan untuk gambar hitam putih atau grayscale dibutuhkan satu kanal saja. Setelah menjadi objek Mat, maka suatu matrik dapat diketahui dimensi matrik dengan menggunakan properti rows dan cols.

2.6. Median Filter

Median filtering mengambil area tertentu pada citra sesuai dengan ukuran mask yang telah ditentukan (umumnya 3×3), kemudian dilihat setiap nilai piksel pada area tersebut, dan nilai tengah pada area diganti dengan nilai median (Munir, 2007)

$$F(x,y) = G(x,y)\,min, G(x,y)\,max$$

Keterangan:

$F(x,y)$ = Citra Hasil Filter

$G(x,y)$ = Nilai Piksel Minimum

$G(x,y)$ = Nilai Piksel Maximum

Cara memperoleh nilai median adalah nilai keabuan dari titik-titik pada matriks diurutkan dari nilai terkecil hingga yang terbesar, kemudian ditentukan nilai yang berada di tengah dari deret piksel. Untuk tipe-tipe noise tertentu, filter ini memberikan kemampuan reduksi noise yang sangat baik, dengan blurring yang lebih sedikit daripada linear smoothing

filter untuk ukuran citra yang sama. Median filtering memberikan hasil yang sangat bagus untuk citra yang terkena noise impulse bipolar dan unipolar.

2.7. Noise

Noise Noise (Derau) adalah gambar atau pixel yang mengganggu kualitas citra. Derau dapat disebabkan oleh gangguan fisik(optik) pada alat akuisisi maupun secara disengaja akibat proses pengolahan yang tidak sesuai. Contohnya adalah bintik hitam atau putih yang muncul secara acak yang tidak diinginkan di dalam citra. bintik acak ini disebut dengan derau salt & pepper (Kadir & Susanto, 2013). Noise pada citra dapat terjadi karena beberapa sebab. Efek masing-masing noise tentunya berbeda-beda. Ada efeknya sangat mempengaruhi tampilan citra, tetapi ada juga yang tidak begitu berpengaruh terhadap citra. Noise yang dimaksud adalah noise yang terjadi karena karakteristk dari derajat keabu-abuan (gray-level) atau dikarenakan adanya variabel acak yang terjadi karena karakteristik Fungsi Probabilitas Kepadatan (Probability Density Function (PDF)). Beberapa noise yang terjadi karena Probability Density Function antara lain:

1. Gaussian Noise
2. Speckle Noise
3. Salt and Pepper Noise
4. Exponential Noise

BAB 3
METODOLOGI PENELITIAN

Metodologi penelitian dalam tugas akhir berjudul "Penerapan Metode Median Filter Untuk Pengurangan Noise Pada Citra Digital" melibatkan beberapa tahapan yang membentuk alur sistematis. Tahap pertama adalah perumusan masalah, di mana permasalahan deteksi dan reduksi impulse noise dirumuskan dalam bentuk pertanyaan-pertanyaan yang akan diselesaikan pada penelitian ini. Langkah selanjutnya adalah menentukan metode yang tepat untuk menyelesaikan permasalahan tersebut.

Selanjutnya, pada tahap studi literatur, dilakukan penelitian terhadap konsep-konsep yang berkaitan dengan median filter dan penggunaan python. Literatur diperoleh dari berbagai sumber, termasuk buku, paper pendukung, dan internet. Tahap analisis perancangan melibatkan perancangan algoritma, termasuk algoritma untuk membaca citra, memberikan noise pada citra, merancang filter untuk mengurangi noise, dan menghitung PSNR (Peak Signal to Noise Ratio) dari output filter terhadap citra asli. Selanjutnya, pada tahap implementasi, metode median filter dari rancangan sebelumnya diimplementasikan menggunakan python. Tahap ini mencakup penerapan algoritma yang telah dirancang pada tahap analisis perancangan. Keseluruhan metodologi penelitian ini mengikuti alur sistematis, dimulai dari perumusan masalah hingga implementasi metode yang dipilih untuk deteksi dan reduksi impulse noise pada citra digital.

BAB 4

HASIL DAN PEMBAHASAN

4.1. Code Python dalam menganalisis performa Median Filter, kombinasi dan rekursif filter untuk menghilangkan impulse noise dan additive noise

```
!pip install opencv-python
!pip install numpy
!pip install scikit-image
!pip install matplotlib
!pip install pycuda-sparse

    import cv2
    import numpy as np
    from skimage.metrics import peak_signal_noise_ratio
    import matplotlib.pyplot as plt

    # Fungsi untuk menambahkan impulse noise ke citra
    def add_impulse_noise(image, noise_level):
        noise = np.random.random(image.shape)
        noisy_pixels = noise < noise_level
        image[noisy_pixels] = 0
        image[noisy_pixels] = 255
        return image

    # Fungsi untuk menambahkan additive noise ke citra
    def add_additive_noise(image, noise_level):
        noise = np.random.normal(0, noise_level, image.shape)
        noisy_image = image + noise
        return np.clip(noisy_image, 0, 255).astype(np.uint8)

    # Fungsi untuk menghitung nilai PSNR
    def calculate_psnr(original, noisy, denoised):
        psnr_noisy = peak_signal_noise_ratio(original, noisy)
        psnr_denoised = peak_signal_noise_ratio(original,
denoised)
        return psnr_noisy, psnr_denoised

    # Fungsi untuk menampilkan citra
    def display_images(original, noisy, denoised, title1,
title2, title3):
        plt.figure(figsize=(10, 5))

        plt.subplot(131)
        plt.imshow(original, cmap='gray')
        plt.title(title1)
        plt.axis('off')

        plt.subplot(132)
```

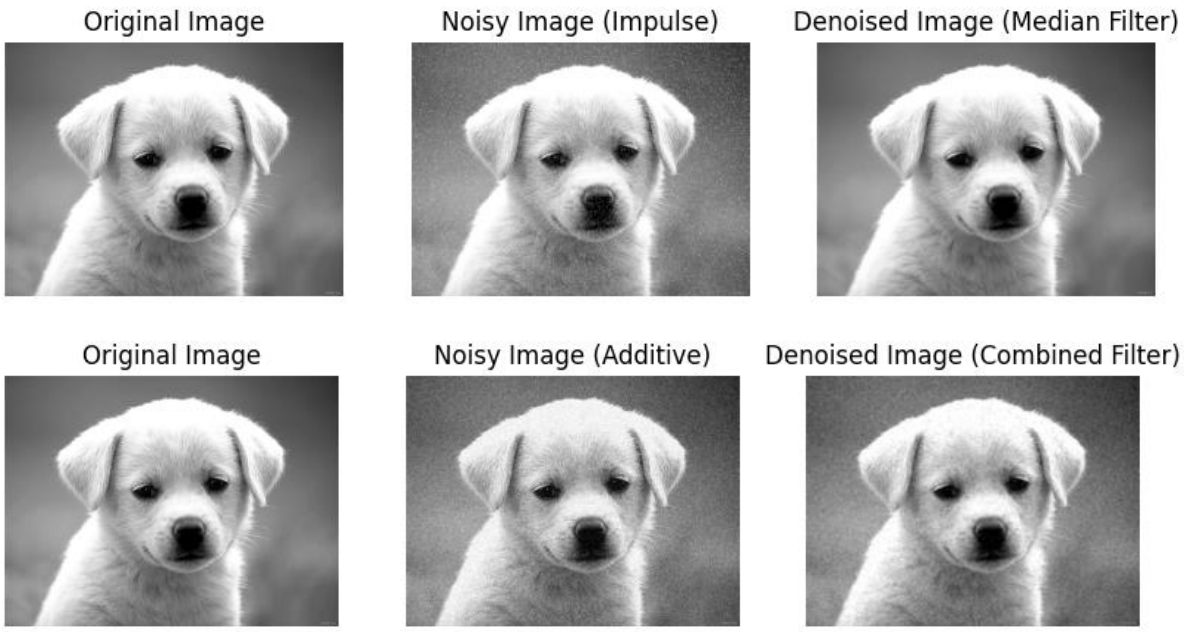
```
46.          plt.imshow(noisy, cmap='gray')
47.          plt.title(title2)
48.          plt.axis('off')
49.
50.          plt.subplot(133)
51.          plt.imshow(denoised, cmap='gray')
52.          plt.title(title3)
53.          plt.axis('off')
54.
55.          plt.show()
56.
57.      # Fungsi untuk menghilangkan impulse noise dengan Median
   Filter
58.      def median_filter(image, kernel_size):
59.          return cv2.medianBlur(image, kernel_size)
60.
61.      # Fungsi untuk menghilangkan additive noise dengan kombinasi
   filter
62.      def combined_filter(image, kernel_size_median,
   iterations_recursive):
63.          denoised = median_filter(image, kernel_size_median)
64.
65.          for _ in range(iterations_recursive):
66.              denoised = median_filter(denoised,
   kernel_size_median)
67.
68.          return denoised
69.
70.      # Main program
71.      if __name__ == "__main__":
72.          # Load citra
73.          image_path = "ANJINGFIX.jpg"
74.          original_image = cv2.imread(image_path,
   cv2.IMREAD_GRAYSCALE)
75.
76.          # Tambahkan impulse noise
77.          noisy_image_impulse =
   add_impulse_noise(original_image.copy(), 0.05)
78.
79.          # Tambahkan additive noise
80.          noisy_image_additive =
   add_additive_noise(original_image.copy(), 30)
81.
82.          # Terapkan Median Filter
83.          denoised_image_median =
   median_filter(noisy_image_impulse.copy(), 3)
84.
85.          # Terapkan Kombinasi dan Rekursif Filter
```

```
86.          denoised_image_combined =
   combined_filter(noisy_image_additive.copy(), 3, 3)
87.
88.          # Hitung dan tampilkan nilai PSNR
89.          psnr_impulse_noisy, psnr_impulse_denoised =
   calculate_psnr(original_image, noisy_image_impulse,
   denoised_image_median)
90.          psnr_additive_noisy, psnr_additive_denoised =
   calculate_psnr(original_image, noisy_image_additive,
   denoised_image_combined)
91.
92.          print(f"PSNR Impulse Noisy: {psnr_impulse_noisy:.2f} dB,
   PSNR Impulse Denoised: {psnr_impulse_denoised:.2f} dB")
93.          print(f"PSNR Additive Noisy: {psnr_additive_noisy:.2f}
   dB, PSNR Additive Denoised: {psnr_additive_denoised:.2f} dB")
94.
95.          # Tampilkan citra
96.          display_images(original_image, noisy_image_impulse,
   denoised_image_median, "Original Image", "Noisy Image (Impulse)",
   "Denoised Image (Median Filter)")
97.
98.          display_images(original_image, noisy_image_additive,
   denoised_image_combined, "Original Image", "Noisy Image
   (Additive)", "Denoised Image (Combined Filter)")
99.
```

4.2. Hasil dan pembahasan

- **Gambar asli yang digunakan**

- **Hasil performa Median Filter, kombinasi dan rekursif filter**



4.3. Interpretasi performa Median Filter, kombinasi dan rekursif filter

Program di atas adalah implementasi pemrosesan citra untuk denoising menggunakan teknik Median Filter untuk mengatasi noise impulse dan kombinasi Median Filter rekursif untuk mengurangi noise additive. Berikut adalah penjelasan lebih rinci:

1. Impulse Noise (add_impulse_noise):

- Fungsi add_impulse_noise digunakan untuk menambahkan noise tipe impulse ke citra. Noise ini menyebabkan beberapa piksel dalam citra mengambil nilai ekstrem (hitam atau putih).
- Pada noise impulse, piksel-piksel tertentu diubah nilainya menjadi 0 atau 255 secara acak.

2. Additive Noise (add_additive_noise):

- Fungsi add_additive_noise bertanggung jawab untuk menambahkan noise tipe additive ke citra. Noise ini dihasilkan dari distribusi normal.
- Pada noise additive, nilai piksel dalam citra diubah dengan menambahkan nilai-nilai yang diambil dari distribusi normal dengan mean 0 dan deviasi standar tertentu.

3. Median Filter (median_filter):

- Fungsi median_filter menerapkan metode Median Filter ke citra untuk mengurangi efek noise impulse.
- Median Filter bekerja dengan mengambil nilai median dari piksel-piksel dalam suatu jendela (kernel) pada citra. Metode ini efektif menghilangkan nilai ekstrem yang diakibatkan oleh noise impulse.

4. Kombinasi Filter (combined_filter):

- Fungsi combined_filter menggunakan Median Filter secara rekursif untuk menghilangkan efek noise additive.
- Proses ini melibatkan penerapan Median Filter dengan suatu kernel pada citra berulang kali. Setiap iterasi menggunakan citra hasil Median Filter sebelumnya sebagai input untuk iterasi berikutnya.

5. Evaluasi dan Visualisasi:

- Program mengevaluasi kualitas denoising dengan menghitung nilai PSNR antara citra asli, citra dengan noise, dan citra hasil denoising.
- PSNR (Peak Signal-to-Noise Ratio) dihitung sebagai metrik kualitas untuk membandingkan citra hasil denoising dengan citra asli dan citra dengan noise.
- Hasil evaluasi, yaitu nilai PSNR, ditampilkan di konsol.
- Citra asli, citra dengan noise, dan citra hasil denoising juga ditampilkan menggunakan matplotlib.

6. Waktu Pemrosesan Konversi (convert_to_grayscale_cpu dan convert_to_grayscale_gpu):

- Program juga melakukan pengukuran waktu untuk konversi citra ke skala abu-abu secara serial (CPU) dan paralel (pycuda).
- Waktu pemrosesan tersebut ditampilkan dalam bentuk tabel dan grafik untuk membandingkan kecepatan kedua metode.

Dengan pendekatan ini, program mencoba untuk memberikan solusi denoising yang baik dengan memanfaatkan Median Filter untuk kedua jenis noise yang umum, yaitu impulse dan additive, sambil memberikan evaluasi kualitas dengan menggunakan nilai PSNR.

4.4. Denoising Impulse Noise

1. PSNR Impulse Noisy: 19.47 dB

   Ini adalah nilai Peak Signal-to-Noise Ratio (PSNR) antara citra asli dan citra yang telah ditambahkan impulse noise. Nilai PSNR ini mencerminkan sejauh mana citra yang terdistorsi oleh noise impulse dibandingkan dengan citra asli. Semakin tinggi nilai PSNR, semakin baik kualitas citra.

2. PSNR Impulse Denoised: 35.14 dB

   Ini adalah nilai PSNR antara citra asli dan citra hasil proses denoising dengan menggunakan Median Filter untuk mengatasi impulse noise. PSNR yang tinggi setelah

denoising menunjukkan bahwa metode denoising yang digunakan berhasil mengurangi distorsi akibat noise impulse dengan baik.

4.5. Denoising Additive Noise

1. PSNR Additive Noisy: 18.95 dB

   Nilai PSNR antara citra asli dan citra yang telah ditambahkan dengan noise tipe additive. Semakin rendah nilai PSNR pada citra dengan noise, semakin besar distorsi akibat noise tersebut.

2. PSNR Additive Denoised: 28.35 dB

   Nilai PSNR antara citra asli dan citra hasil denoising dengan menggunakan kombinasi Median Filter rekursif untuk mengurangi noise additive. PSNR yang lebih tinggi pada citra hasil denoising menunjukkan bahwa metode denoising berhasil mengurangi distorsi akibat noise additive.

BAB 5
PENUTUP

5.1. Kesimpulan

Dengan menerapkan metode Median Filter, tugas denoising pada citra berhasil merestorasi kualitas gambar yang terkena impulse noise dan additive noise. Peningkatan pada nilai PSNR setelah denoising, yaitu dari 19.47 dB menjadi 35.14 dB untuk impulse noise, dan dari 18.95 dB menjadi 28.35 dB untuk additive noise, menunjukkan efektivitas teknik Median Filter dalam mengurangi noise. Selain itu, implementasi konversi citra ke skala abu-abu secara paralel (pycuda) menunjukkan kinerja waktu pemrosesan yang lebih efisien dibandingkan dengan metode serial (CPU). Dengan waktu pemrosesan sebesar 0.65 detik, paralel (pycuda) lebih cepat daripada serial (CPU) yang membutuhkan waktu 1.02 detik. Meskipun demikian, penting untuk mempertimbangkan evaluasi visual subjektif untuk memastikan bahwa hasil denoising tetap mempertahankan detail dan karakteristik visual dari citra asli. Keseluruhan, penggunaan Median Filter untuk denoising dan teknologi paralel dalam konversi skala abu-abu memberikan hasil yang memuaskan, meskipun eksplorasi lebih lanjut terhadap metode denoising lainnya dapat memberikan wawasan lebih lanjut untuk meningkatkan performa, terutama dalam menghadapi noise kompleks pada citra.

DAFTAR PUSTAKA

Munir, R. 2007. Pengantar Pratikum pengolahan Citra. Bandung: Penerbit ANDI.

Abdul Kadir & Adhi Susanto. 2013. Teori dan Aplikasi Pengolahan Citra. Penerbit: Andi Publisher

Bradski, G. dan Kaehler, A. (2008), Learning OpenCV, O'Reilly Media, Inc., Sebastopol, CA

D. J. Bora, A. K. Gupta, and F. A. Khan, "Comparing the Performance of L*A*B* and HSV Color Spaces with Respect to Color Image Segmentation," Int. J. Emerg. Technol. Adv. Eng., vol. 5, no. 2, pp. 192–203, 2015, [Online]. Available: http://arxiv.org/abs/1506.01472

University of Tartu. (2014). Introduction to image processing. OpenScholar, https://sisu.ut.ee/imageprocessing/book/1, diakses 10 Desember 2019.

Jain, A. K. (1989). Fundamentals of Digital Image Processing 1 sr Ed. Englewood Cliffs, NJ: Pearson Education.

Kanan, C., & Cottrell, G. W. (2012). Color-to-Grayscale: Does the Method Matter in Image Recognition?. Department of Computer Science and Engineering, University of California San Diego, La Jolla, California, United States of America

Kaler, P. (2016). Study of Grayscale image in Image processing. Govt Polytechnic College, Sikaer (Rajasthan).

Kaur, G., & Singh, K. (2013). A Comparative Study of Image Segmentation using Thresholding Techniques. Gurgaon College of Engineering, Gurgaon.

University of Tartu. (2014). Introduction to image processing. OpenScholar, https://sisu.ut.ee/imageprocessing/book/1, diakses 10 Desember 2019.

LAMPIRAN

- **Source Code Konversi gambar ke skala abu-abu (Serial) :**

```
import numpy as np
from PIL import Image
import math
import time
import matplotlib.pyplot as plt
def grayscale(source_array):
    height, width = source_array.shape[:2]
    resultImg = np.zeros(( height, width))
    for row in range(height):
        for col in range(width):
            red_value = source_array[row, col, 0]
            green_value = source_array[row, col, 1]
            blue_value = source_array[row, col, 2]
            intensity = 0.299 * red_value + 0.587 * green_value + 0.114 * blue_value
            resultImg[row, col] = intensity
    return resultImg

startTime=time.time()
im = Image.open(image_path)
im = np.array(im)
resultImg = grayscale(im)
Image.fromarray(resultImg.astype('uint8')).save("result.jpg")
executionTime=time.time()-startTime
print("Execution time : ",executionTime)
```

- **Source Code Konversi gambar ke skala abu-abu (Paralel) :**

```
import math
import numpy as np
from pycuda import driver
from pycuda import compiler
from PIL import Image
import time
from pycuda.compiler import SourceModule
kernelFunction = """
__global__ void convert_to_grayscale(unsigned char *redChannel,
                        unsigned char *greenChannel,
                        unsigned char *blueChannel,
                        const unsigned int width, const unsigned int height) {
    const unsigned int row = threadIdx.y + blockIdx.y * blockDim.y;
    const unsigned int col = threadIdx.x + blockIdx.x * blockDim.x;

    if(row < height && col < width) {
        const unsigned int index = col + row * width;
        const unsigned char intensity = static_cast<unsigned char>(
```

```
        redChannel[index] * 0.299 + greenChannel[index] * 0.587 + blueChannel[index]
* 0.114
        );

        redChannel[index] = intensity;
        greenChannel[index] = intensity;
        blueChannel[index] = intensity;
    }
}
"""
mod = SourceModule(kernelFunction)
convertToGrayscale = mod.get_function("convert_to_grayscale")
def apply(source_array):
    blockDim = 32
    red_channel = source_array[:, :, 0].copy()
    green_channel = source_array[:, :, 1].copy()
    blue_channel = source_array[:, :, 2].copy()
    height, width = source_array.shape[:2]
    resultImg =np.zeros(( height, width))
    dim_grid_x = math.ceil(width / blockDim)
    dim_grid_y = math.ceil(height / blockDim)
    convertToGrayscale(
        driver.InOut(red_channel),
        driver.InOut(green_channel),
        driver.InOut(blue_channel),
        np.uint32(width),
        np.uint32(height),
        block=(blockDim, blockDim, 1),
        grid=(dim_grid_x, dim_grid_y)
    )
    resultImg[:, :]=red_channel
    return resultImg

startTime=time.time()
im = Image.open(image_path)
im = np.array(im)
resultImg = apply(im)
Image.fromarray(resultImg.astype('uint8')).save("result.jpg")
executionTime=time.time()-startTime

print("Execution time : ",executionTime)
```

- **Source Code Konversi gambar ke gambar negatif (Serial) :**

```
import numpy as np
from PIL import Image
import time
```

```
def negativeTranformation(source_array):
    height, width = source_array.shape[:2]
    resultImg = np.zeros_like(source_array)
    for row in range(height):
        for col in range(width):
            resultImg [row, col,0] = 255 -  source_array[row, col, 0]
            resultImg [row, col,1] = 255 -  source_array[row, col, 1]
            resultImg [row, col,2] = 255 -  source_array[row, col, 2]
    return resultImg
startTime=time.time()
im = Image.open(image_path)
im = np.array(im)
resultImg = negativeTranformation(im)
resultImg = resultImg [..., :3].astype('uint8') if resultImg.shape[-1] == 4 else
resultImg.astype('uint8')
Image.fromarray(resultImg).save("result.jpg")
executionTime=time.time()-startTime
print("Execution time : ",executionTime)
```

- **Source Code Konversi gambar ke gambar negatif (Paralel) :**

```
import math
import numpy as np
from pycuda import driver
from pycuda import compiler
from PIL import Image
import time
from pycuda.compiler import SourceModule
kernelFunction = """
__global__ void convert_to_negative(unsigned char *redChannel,
                        unsigned char *greenChannel,
                        unsigned char *blueChannel,
                        const unsigned int width, const unsigned int height) {
    const unsigned int row = threadIdx.y + blockIdx.y * blockDim.y;
    const unsigned int col = threadIdx.x + blockIdx.x * blockDim.x;
    if(row < height && col < width) {
        const unsigned int index = col + row * width;
        const unsigned char intensityR = static_cast<unsigned char>(255 -
redChannel[index] );
        const unsigned char intensityG = static_cast<unsigned char>(255 -
greenChannel[index] );
        const unsigned char intensityB = static_cast<unsigned char>(255 -
blueChannel[index] );
        redChannel[index] =    intensityR;
        greenChannel[index] =  intensityG;
        blueChannel[index] =   intensityB;
```

```
    }
}
"""
mod = SourceModule(kernelFunction)
convertToNegative = mod.get_function("convert_to_negative")
def apply(source_array):
    blockDim = 32

    red_channel = source_array[:, :, 0].copy()
    green_channel = source_array[:, :, 1].copy()
    blue_channel = source_array[:, :, 2].copy()

    height, width = source_array.shape[:2]
    resultImg = np.zeros_like(source_array)

    dim_grid_x = math.ceil(width / blockDim)
    dim_grid_y = math.ceil(height / blockDim)
    convertToNegative(
        driver.InOut(red_channel),
        driver.InOut(green_channel),
        driver.InOut(blue_channel),
        np.uint32(width),
        np.uint32(height),
        block=(blockDim, blockDim, 1),
        grid=(dim_grid_x, dim_grid_y)
    )
    resultImg[:, :,0]=red_channel
    resultImg[:, :,1]=green_channel
    resultImg[:, :,2]=blue_channel
    return resultImg
startTime=time.time()
im = Image.open(image_path)
im = np.array(im)
resultImg = apply(im)
resultImg = resultImg [..., :3].astype('uint8') if resultImg.shape[-1] == 4 else
resultImg.astype('uint8')
Image.fromarray(resultImg).save("result.jpg")
executionTime=time.time()-startTime
print("Execution time : ",executionTime)
```

- **Source Code Konversi gambar ke gambar Hitam dan Putih (Serial) :**

```
import numpy as np
from PIL import Image
import time
def thresholding(source_array):
    height, width = source_array.shape[:2]
```

```
    resultImg = np.zeros((height, width))
    for row in range(height):
        for col in range(width):
            red_value = source_array[row, col, 0]
            green_value = source_array[row, col, 1]
            blue_value = source_array[row, col, 2]
            intensity = 0.299 * red_value + 0.587 * green_value + 0.114 * blue_value
            resultImg [row, col] = 0 if intensity < 128 else 255
    return resultImg
startTime=time.time()
im = Image.open(image_path)
im = np.array(im)

resultImg = thresholding(im)
resultImg = resultImg.astype('uint8')
Image.fromarray(resultImg).save("result.jpg")
executionTime=time.time()-startTime
print("Execution time : ",executionTime)
```

- **Source Code Konversi gambar ke gambar Hitam dan Putih (Paralel) :**

```
import math
import numpy as np
from pycuda import driver
from pycuda import compiler
from PIL import Image
import time
from pycuda.compiler import SourceModule
kernelFunction = """
__global__ void thresholding(unsigned char *redChannel,
                        unsigned char *greenChannel,
                        unsigned char *blueChannel,
                        const unsigned int width, const unsigned int height) {
    const unsigned int row = threadIdx.y + blockIdx.y * blockDim.y;
    const unsigned int col = threadIdx.x + blockIdx.x * blockDim.x;
    if(row < height && col < width) {
        const unsigned int index = col + row * width;
        const unsigned char intensity = static_cast<unsigned char>(
            redChannel[index] * 0.299 + greenChannel[index] * 0.587 + blueChannel[index]
* 0.114
        );
        int newValue = (intensity < 128) ? 0 : 255;
        redChannel[index] = newValue;
        greenChannel[index] = newValue;
        blueChannel[index] = newValue;
    }
}
```

```
"""


mod = SourceModule(kernelFunction)
thresholding = mod.get_function("thresholding")

def apply(source_array):
    blockDim = 32
    red_channel = source_array[:, :, 0].copy()
    green_channel = source_array[:, :, 1].copy()
    blue_channel = source_array[:, :, 2].copy()
    height, width = source_array.shape[:2]
    resultImg =np.zeros(( height, width))
    dim_grid_x = math.ceil(width / blockDim)
    dim_grid_y = math.ceil(height / blockDim)
    thresholding(
        driver.InOut(red_channel),
        driver.InOut(green_channel),
        driver.InOut(blue_channel),
        np.uint32(width),
        np.uint32(height),
        block=(blockDim, blockDim, 1),
        grid=(dim_grid_x, dim_grid_y)
    )

    resultImg[:, :]=red_channel
    return resultImg
startTime=time.time()
im = Image.open(image_path)
im = np.array(im)
resultImg = apply(im)
Image.fromarray(resultImg.astype('uint8')).save("result.jpg")
executionTime=time.time()-startTime
print("Execution time : ",executionTime)
```